



МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
ІМЕНІ ІГОРЯ СІКОРСЬКОГО”

Факультет прикладної математики
Кафедра програмного забезпечення комп’ютерних систем

Лабораторна робота №3
з дисципліни «Бази даних»
тема «Засоби оптимізації роботи СУБД PostgreSQL»

Виконав(ла)
студент(ка) II курсу
групи КП-03

Сітайло Анна Сергіївна
(*прізвище, ім’я, по батькові*)

Варіант №17

Перевірів
“ ____ ” “ ____ ” 20__ р.
викладач

Радченко Костянтин
Олександрович
(*прізвище, ім’я, по батькові*)

Київ 2021

Мета

Здобуття практичних навичок використання засобів оптимізації СУБД PostgreSQL.

Завдання

1. Перетворити модуль “Модель” з шаблону MVC лабораторної роботи №2 у вигляд об’єктно-реляційної проєкції (ORM).
2. Створити та проаналізувати різні типи індексів у PostgreSQL: GIN, BRIN.
3. Розробити тригер бази даних PostgreSQL з умовою «before update, delete».

Результати

1.

```
import random
import psycopg2

class Student:
    def __init__(self, connection, students):
        self.connection = connection
        self.students = students

    def add_student(self, line):
        try:
            line_inserting = line.split(", ")
            insert_query = self.students.insert().values(
                name=f'{line_inserting[0]}',
                age=line_inserting[1],
                grade=line_inserting[2]
            )
            self.connection.execute(insert_query)
        except:
            print("Error: student was not added!")

    def update_student(self, line):
        try:
            line_updating = line.split(", ")
            update_query = self.students.update().where(self.students.c.id == line_updating[0]).values(
                name=f'{line_updating[1]}',
                age=line_updating[2],
                grade=line_updating[3]
            )
```

```

    )
    self.connection.execute(update_query)
except:
    print("Error: student was not updated!")

def delete_student(self, line_deleting):
    try:
        delete_query = self.students.delete().where(self.students.c.id == line_deleting)
        self.connection.execute(delete_query)
    except:
        print("Error: student was not deleted")

def generate_random_students(self, number_of_lines):
    try:
        i = 0
        while i < number_of_lines:
            insert_query = self.students.insert().values(
                name=f'{chr(random.randint(65, 90))}{chr(random.randint(65, 90))}',
                age=random.randint(5, 18),
                grade=random.randint(1, 11)
            )
            self.connection.execute(insert_query)
            i += 1
    except:
        print("Error: student was not generated!")

```

```
class Teacher:
    def __init__(self, connection, teachers):
        self.connection = connection
        self.teachers = teachers

    def add_teacher(self, line):
        try:
            line_inserting = line.split(", ")
            insert_query = self.teachers.insert().values(
                name=f'{line_inserting[0]}',
                age=line_inserting[1],
                work_experience=line_inserting[2]
            )
            self.connection.execute(insert_query)
        except:
            print("Error: teacher was not added!")

    def update_teacher(self, line):
        try:
            line_updating = line.split(", ")
            update_query = self.teachers.update().where(self.teachers.c.id == line_updating[0]).values(
                name=f'{line_updating[1]}',
                age=line_updating[2],
                work_experience=line_updating[3]
            )
            self.connection.execute(update_query)
        except:
            print("Error: teacher was not updated!")
```

```

def delete_teacher(self, line_deleting):
    try:
        delete_query = self.teachers.delete().where(self.teachers.c.id == line_deleting)
        self.connection.execute(delete_query)
    except:
        print(f"Error: teacher with id = {line_deleting} does not exist!")

def generate_random_teachers(self, number_of_lines):
    try:
        i = 0
        while i < number_of_lines:
            insert_query = self.teachers.insert().values(
                name=f'{chr(random.randint(65, 90))}{chr(random.randint(65, 90))}',
                age=random.randint(25, 75),
                work_experience=random.randint(0, 50)
            )
            self.connection.execute(insert_query)
            i += 1
    except:
        print("Error: teacher was not generated!")

```

```

class Subject:
    def __init__(self, connection, subjects):
        self.connection = connection

```

```

class Subject:
    def __init__(self, connection, subjects):
        self.connection = connection
        self.subjects = subjects

    def create_trigger(self):
        con = psycopg2.connect(
            database='postgres',
            user='postgres',
            password=1234567890,
            host='localhost',
            port=5432,
        )
        create_trigger = """CREATE TRIGGER t_subject
AFTER UPDATE OR DELETE ON subjects FOR EACH ROW EXECUTE PROCEDURE add_to_log ();"""
        con.autocommit = True
        cursor = con.cursor()
        cursor.execute(create_trigger)

    def add_subject(self, line):
        try:
            line_inserting = line.split(", ")
            insert_query = self.subjects.insert().values(
                name=f'{line_inserting[0]}',
                classes_per_semester=line_inserting[1]
            )

```

```

def update_subject(self, line):
    try:
        line_updating = line.split(", ")
        update_query = self.subjects.update().where(self.subjects.c.id == line_updating[0]).values(
            name=f'{line_updating[1]}',
            classes_per_semester=line_updating[2]
        )
        self.create_trigger()
        self.connection.execute(update_query)
    except:
        print("Error: subject was not updated!")

def delete_subject(self, line_deleting):
    try:
        delete_query = self.subjects.delete().where(self.subjects.c.id == line_deleting)
        # self.create_trigger()
        self.connection.execute(delete_query)
    except:
        print(f"Error: subject with id = {line_deleting} does not exist!")

def generate_random_subjects(self, number_of_lines):
    try:
        i = 0
        subject_name = ["literature", "math", "science", "art", "biology", "PE", "english", "french", "geography",
                        "history"]
        while i < number_of_lines:
            insert_query = self.subjects.insert().values(

```

```

        name=f'{subject_name[random.randint(0, 9)]}',
        classes_per_semester=random.randint(8, 80)
    )
    self.connection.execute(insert_query)
    i += 1
except:
    print("Error: subject was not generated!")

class Subject_Teacher:
    def __init__(self, connection, subjects_teachers):
        self.connection = connection
        self.subjects_teachers = subjects_teachers

    def add_subjects_teachers_record(self, line):
        try:
            line_inserting = line.split(", ")
            insert_query = self.subjects_teachers.insert().values(
                subject_id=line_inserting[0],
                teacher_id=line_inserting[1]
            )
            self.connection.execute(insert_query)
        except:
            print("Error: record was not added!")

    def update_subjects_teachers_record(self, line):

```



```
def update_subjects_teachers_record(self, line):
    try:
        line_updating = line.split(" ")
        update_query = self.subjects_teachers.update().where(self.subjects_teachers.c.id == line_updating[0]).values(
            subject_id=line_updating[1],
            teacher_id=line_updating[2]
        )
        self.connection.execute(update_query)
    except:
        print("Error: record was not updated!")

def delete_subjects_teachers_record(self, line_deleting):
    try:
        delete_query = self.subjects_teachers.delete().where(self.subjects_teachers.c.id == line_deleting)
        self.connection.execute(delete_query)
    except:
        print(f"Error: record with id = {line_deleting} does not exist!")

def generate_random_subjects_teachers_records(self, number_of_lines):
    try:
        i = 0
        while i < number_of_lines:
            insert_query = self.subjects_teachers.insert().values(
                subject_id=random.randint(1, 500),
                teacher_id=random.randint(1, 500)
            )
            self.connection.execute(insert_query)
            i += 1
```

```

        teacher_id=random.randint(1, 500)
    )
    self.connection.execute(insert_query)
    i += 1
except:
    print("Error: record was not generated!")

```

```

class Schedule:
    def __init__(self, connection, schedule):
        self.connection = connection
        self.schedule = schedule

    def add_schedule_record(self, line):
        try:
            line_inserting = line.split(", ")
            insert_query = self.schedule.insert().values(
                day=f'{line_inserting[0]}',
                time=f'{line_inserting[1]}',
                subject_teacher_id=line_inserting[2],
                student_id=line_inserting[3]
            )
            self.connection.execute(insert_query)
        except:
            print("Error: record was not added!")

```

```

def update_schedule_record(self, line):
    try:
        line_updating = line.split(", ")
        update_query = self.schedule.update().where(self.schedule.c.id == line_updating[0]).values(
            day=f'{line_updating[1]}',
            time=f'{line_updating[2]}',
            subject_teacher_id=line_updating[3],
            student_id=line_updating[4]
        )
        self.connection.execute(update_query)
    except:
        print("Error: record was not updated!")

def delete_schedule_record(self, line_deleting):
    try:
        delete_query = self.schedule.delete().where(self.schedule.c.id == line_deleting)
        self.connection.execute(delete_query)
    except:
        print(f"Error: record with id = {line_deleting} does not exist!")

def generate_random_schedule_records(self, number_of_lines):
    try:
        i = 0
        days_of_week = ["monday", "tuesday", "wednesday", "thursday", "friday", "saturday"]
        while i < number_of_lines:
            insert_query = self.schedule.insert().values(

```

```

        day=days_of_week[random.randint(0, 5)],
        time=f'{random.randint(8, 16)}:{random.randint(0, 59)}',
        subject_teacher_id=random.randint(1, 500),
        student_id=random.randint(2, 500)
    )
    self.connection.execute(insert_query)
    i += 1
except:
    print("Error: record was not generated!")

class Mark:
    def __init__(self, connection, marks):
        self.connection = connection
        self.marks = marks

    def add_mark(self, line):
        try:
            line_inserting = line.split(", ")
            insert_query = self.marks.insert().values(
                student_id=line_inserting[0],
                subject_teacher_id=line_inserting[1],
                points=line_inserting[2]
            )
            self.connection.execute(insert_query)

```

```

    def generate_random_marks(self, number_of_lines):
        try:
            i = 0
            while i < number_of_lines:
                insert_query = self.marks.insert().values(
                    student_id=random.randint(2, 500),
                    subject_teacher_id=random.randint(1, 500),
                    points=random.randint(1, 12)
                )
                self.connection.execute(insert_query)
                i += 1
        except:
            print("Error: marks were not generated!")

```

```

class Search:

    def __init__(self, connection, students, teachers, subjects, subjects_teachers, schedule, marks):
        self.connection = connection
        self.students = students
        self.teachers = teachers
        self.subjects = subjects
        self.subjects_teachers = subjects_teachers
        self.schedule = schedule
        self.marks = marks

    def find_subjects_teachers_records_by_name_classes_age(self, classes_range, subject_name, age_range):
        try:
            find_query = select([self.subjects.c.name,
                                self.subjects.classes_per_semester,
                                self.teachers.name,
                                self.teachers.age]).where(and_(
                                    self.subjects_teachers.c.subject_id == self.subjects.c.id,
                                    self.subjects.c.name == f'{subject_name}',
                                    self.subjects.c.classes_per_semester > {classes_range[0]},
                                    self.subjects.c.classes_per_semester < {classes_range[1]},
                                    self.teachers.c.age > {age_range[0]},
                                    self.teachers.c.age < {age_range[1]}))

            result = self.connection.execute(find_query)

            for line in result.fetchall():

```

```

        print(line)
    except:
        print("Error: records were not found!")

def find_schedule_records_by_id_name_day(self, id_range, teacher_name, day):
    try:
        find_query = select([self.schedule.c.id,
                               self.schedule.c.day,
                               self.schedule.c.time,
                               self.subjects.c.name,
                               self.teachers.c.name,
                               self.students.c.name]).where(and_(
                                    self.schedule.c.id > id_range[0],
                                    self.schedule.c.id < id_range[1],
                                    self.subjects_teachers.id == self.schedule.c.subject_teacher_id,
                                    self.subjects.c.id == self.subjects_teachers.c.subject_id,
                                    self.teachers.c.id == self.subjects_teachers.c.teacher_id,
                                    self.students.c.id == self.schedule.c.student_id,
                                    self.teachers.c.name == f'{teacher_name}',
                                    self.schedule.c.day == f'{day}'))

        result = self.connection.execute(find_query)

        for line in result.fetchall():
            print(line)
    except:
        print("Error: records were not found!")

```

```

def find_marks_records_by_name_mark_grade(self, mark_range, grade, subject_name):
    try:
        find_query = select([self.marks.c.points,
                               self.students.c.name,
                               self.students.c.grade,
                               self.subjects.c.name,
                               self.teachers.c.name]).where(and_(
                                    self.marks.c.points >= mark_range[0],
                                    self.marks.c.points <= mark_range[1],
                                    self.students.c.id == self.marks.c.student_id,
                                    self.subjects_teachers.c.id == self.marks.c.subject_teacher_id,
                                    self.teachers.c.id == self.subjects_teachers.c.teacher_id,
                                    self.students.c.grade == grade,
                                    self.subjects.c.name == f'{subject_name}'))

        result = self.connection.execute(find_query)

        for line in result.fetchall():
            print(line)
    except:
        print("Error: records were not found!")

```

```

except:
    print("Error: mark was not added!")

def update_mark(self, line):
    try:
        line_updating = line.split(", ")
        update_query = self.marks.update().where(self.marks.c.id == line_updating[0]).values(
            student_id=line_updating[1],
            subject_teacher_id=line_updating[2],
            points=line_updating[3]
        )
        self.connection.execute(update_query)
    except:
        print("Error: mark was not updated!")

def delete_mark(self, line_deleting):
    try:
        delete_query = self.marks.delete().where(self.marks.c.id == line_deleting)
        self.connection.execute(delete_query)
    except:
        print(f"Error: mark with id = {line_deleting} does not exist!")

def generate_random_marks(self, number_of_lines):
    try:
        i = 0
        while i < number_of_lines:
            insert_query = self.marks.insert().values(

```

```
meta = MetaData()

def create_connection():
    connection = psycopg2.connect(
        database='lab3',
        user='postgres',
        password=1234567890,
        host='localhost',
        port=5432,
    )
    return connection

students = Table('students', meta,
    Column('id', Integer, nullable=False, primary_key=True),
    Column('name', String(250), nullable=False),
    Column('age', Integer, nullable=False),
    Column('grade', Integer, nullable=False)
)

teachers = Table('teachers', meta,
    Column('id', Integer, nullable=False, primary_key=True),
    Column('name', String(250), nullable=False),
    Column('age', Integer, nullable=False),
    Column('work_experience', Integer, nullable=False)
)

subjects = Table('subjects', meta,
    Column('id', Integer, nullable=False, primary_key=True),
```



```

subjects = Table('subjects', meta,
                  Column('id', Integer, nullable=False, primary_key=True),
                  Column('name', String(250), nullable=False),
                  Column('classes_per_semester', Integer, nullable=False)
                  )

subjects_teachers = Table('subjects_teachers', meta,
                           Column('id', Integer, nullable=False, primary_key=True),
                           Column('subject_id', Integer, ForeignKey('subjects.id'), nullable=False),
                           Column('teacher_id', Integer, ForeignKey('teachers.id'), nullable=False)
                           )

schedule = Table('schedule', meta,
                  Column('id', Integer, nullable=False, primary_key=True),
                  Column('day', String(250), nullable=False),
                  Column('time', String(250), nullable=False),
                  Column('subject_teacher_id', Integer, ForeignKey('subjects_teachers.id'), nullable=False),
                  Column('student_id', Integer, ForeignKey('students.id'), nullable=False)
                  )

marks = Table('marks', meta,
               Column('id', Integer, nullable=False, primary_key=True),
               Column('student_id', Integer, ForeignKey('students.id'), nullable=False),
               Column('subject_teacher_id', Integer, ForeignKey('subjects_teachers.id'), nullable=False),
               Column('points', Integer, nullable=False)
               )

```

2.

```

def get_index(self):
    query = "CREATE INDEX ON students"
    self.connection.execute(query)

```

```

def get_index(self):
    query = "CREATE INDEX ON teachers"
    self.connection.execute(query)

```

```

def get_index(self):
    query = "CREATE INDEX ON subjects"
    self.connection.execute(query)

```

```

def get_index(self):
    query = "CREATE INDEX ON subjects_teachers"
    self.connection.execute(query)

```

```
def get_index(self):  
    query = "CREATE INDEX ON schedule"  
    self.connection.execute(query)
```

```
def get_index(self):  
    query = "CREATE INDEX ON marks"  
    self.connection.execute(query)
```






Деякі індекси націлюються на якість виконання запитів, жертвуючи часом, а деякі першим пріоритетом ставлять саме швидкість виконання. Індекси потрібно використовуватизалежно від бажаної мети та виду даних.

```

1 SELECT * FROM public.students
2 ORDER BY id ASC

```

Data Output Explain Messages Notifications

		id [PK] integer 	name character varying (250) 	age integer 	grade integer 
1		2	XB	11	8
2		3	PP	11	11
3		4	IH	11	9
4		5	PG	15	7
5		6	KN	10	7
6		7	MJ	14	7
7		8	MC	9	4
8		9	CI	10	8
9		10	LG	9	9
10		11	XV	10	6
11		12	GE	17	5
12		13	SL	14	1
13		14	QC	11	10
14		15	CS	9	9

3.





```
1 SELECT * FROM public.teachers
2 ORDER BY id ASC
```

Data Output Explain Messages Notifications

	id [PK] integer	name character varying (250)	age integer	work_experience integer
1	1	AX	66	29
2	2	ZM	70	32
3	3	ZW	72	28
4	4	ED	65	47
5	5	BG	43	8
6	6	GH	67	32
7	7	HU	55	27
8	8	RR	62	33
9	9	DT	68	38
10	10	HH	26	16
11	11	MB	31	19
12	12	LT	51	38
13	13	QO	33	1
14	14	EH	50	48





```
1 SELECT * FROM public.subjects
2 ORDER BY id ASC
```

Data Output Explain Messages Notifications

	 id [PK] integer 	name character varying (250) 	classes_per_semester integer 
1	1	biology	8
2	2	math	24
3	3	english	71
4	4	art	48
5	5	biology	12
6	6	geography	23
7	7	PE	18
8	8	history	23
9	9	PE	60
10	10	history	77
11	11	literature	27
12	12	math	15
13	13	geography	25
14	14	art	44

```
1 SELECT * FROM public.subjects_teachers
2 ORDER BY id ASC
```

Data Output Explain Messages Notifications

	 id [PK] integer 	subject_id integer 	teacher_id integer 	
1	1	411	317	
2	2	444	429	
3	3	459	116	
4	4	176	158	
5	5	13	18	
6	6	421	408	
7	7	20	307	
8	8	375	96	
9	9	411	466	
10	10	132	296	
11	11	52	191	
12	12	343	432	
13	13	8	106	
14	14	63	293	

```

1 SELECT * FROM public.schedule
2 ORDER BY id ASC






```

Data Output Explain Messages Notifications

	id [PK] integer	day character varying (250)	time character varying (250)	subject_teacher_id integer	student_id integer
1	1	saturday	9:42	484	49
2	2	thursday	9:9	364	66
3	3	thursday	12:41	105	318
4	4	friday	8:21	166	393
5	5	wednesday	16:13	458	164
6	6	monday	9:14	436	397
7	7	saturday	8:21	351	333
8	8	tuesday	16:57	50	6
9	9	friday	9:1	341	221
10	10	tuesday	9:25	470	62
11	11	saturday	12:34	322	402
12	12	monday	13:31	371	400
13	13	friday	10:44	199	488
14	14	tuesday	16:58	345	139

```
1 SELECT * FROM public.marks
2 ORDER BY id ASC
```

Data Output Explain Messages Notifications

	 id [PK] integer 	student_id integer 	subject_teacher_id integer 	points integer 
1	4	255	373	4
2	5	367	391	9
3	6	472	458	6
4	7	205	236	8
5	8	171	268	6
6	9	195	454	7
7	10	255	434	9
8	11	132	11	9
9	12	175	34	9
10	13	447	447	3
11	14	137	5	9
12	15	215	264	8
13	16	405	349	3
14	17	14	431	11


```
def create_trigger(self):
    con = psycopg2.connect(
        database='postgres',
        user='postgres',
        password=1234567890,
        host='localhost',
        port=5432,
    )
    create_trigger = """CREATE TRIGGER t_student
BEFORE UPDATE OR DELETE ON students FOR EACH ROW EXECUTE PROCEDURE add_to_log ();"""
    con.autocommit = True
    cursor = con.cursor()
    cursor.execute(create_trigger)
```

```
def create_trigger(self):
    con = psycopg2.connect(
        database='postgres',
        user='postgres',
        password=1234567890,
        host='localhost',
        port=5432,
    )
    create_trigger = """CREATE TRIGGER t_teacher
BEFORE UPDATE OR DELETE ON teachers FOR EACH ROW EXECUTE PROCEDURE add_to_log ();"""
    con.autocommit = True
    cursor = con.cursor()
    cursor.execute(create_trigger)
```

```

def create_trigger(self):
    con = psycopg2.connect(
        database='postgres',
        user='postgres',
        password=1234567890,
        host='localhost',
        port=5432,
    )
    create_trigger = """CREATE TRIGGER t_subject
    BEFORE UPDATE OR DELETE ON subjects FOR EACH ROW EXECUTE PROCEDURE add_to_log ();"""
    con.autocommit = True
    cursor = con.cursor()
    cursor.execute(create_trigger)

```

```

def create_trigger(self):
    con = psycopg2.connect(
        database='postgres',
        user='postgres',
        password=1234567890,
        host='localhost',
        port=5432,
    )
    create_trigger = """CREATE TRIGGER t_subject_teacher
    BEFORE UPDATE OR DELETE ON subjects_teachers FOR EACH ROW EXECUTE PROCEDURE add_to_log ();"""
    con.autocommit = True
    cursor = con.cursor()
    cursor.execute(create_trigger)

```

```

def create_trigger(self):
    con = psycopg2.connect(
        database='postgres',
        user='postgres',
        password=1234567890,
        host='localhost',
        port=5432,
    )
    create_trigger = """CREATE TRIGGER t_schedule
    BEFORE UPDATE OR DELETE ON schedule FOR EACH ROW EXECUTE PROCEDURE add_to_log ();"""
    con.autocommit = True
    cursor = con.cursor()
    cursor.execute(create_trigger)

```

```

def create_trigger(self):
    con = psycopg2.connect(
        database='postgres',
        user='postgres',
        password=1234567890,
        host='localhost',
        port=5432,
    )
    create_trigger = """CREATE TRIGGER t_mark
    BEFORE UPDATE OR DELETE ON marks FOR EACH ROW EXECUTE PROCEDURE add_to_log ();"""
    con.autocommit = True
    cursor = con.cursor()
    cursor.execute(create_trigger)

```

Висновки

Виконуючи лабораторну роботу №3 ми навчилися оптимізовувати СУБД PostgreSQL. А саме: приводити код з SQL-запитами у вигляд об'єктно-реляційної проєкції (ORM), створювати та аналізувати різні типи індексів у PostgreSQL та розроблювати тригери бази даних.